**NAME:A.KARTHIKEYAN**

**REG NO:192325143**

**SUB:CSA0814-PYTHON PROGRAM FOR LIST IMPLEMENTATION**

**TITLE 3 : Real-Time Traffic Monitoring System**

**Scenario:**

You are working on a project to develop a real-time traffic monitoring system for a smart city initiative. The system should provide real-time traffic updates and suggest alternative routes.

**Tasks:**

1. **Model the data flow for fetching real-time traffic information from an external API and displaying it to the user.**
2. **Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.**
3. **Display current traffic conditions, estimated travel time, and any incidents or delays.**
4. **Allow users to input a starting point and destination to receive traffic updates and alternative routes.**

**Deliverables:**

* Data flow diagram illustrating the interaction between the application and the API.
* Pseudocode and implementation of the traffic monitoring system.
* Documentation of the API integration and the methods used to fetch and display traffic data.
* Explanation of any assumptions made and potential improvements.

**Approach**

To create a real-time traffic monitoring system using Tkinter and the Google Maps Distance Matrix API, we aim to develop a graphical user interface (GUI) application that allows users to input origin and destination addresses and fetches the traffic data for those routes. The application will display the distance, duration, and traffic status of the requested routes.

**Pseudocode**

1. **Import Libraries**
   * Import necessary libraries (requests, json, tkinter, PIL).
2. **Define Functions**
   * get\_traffic\_data(origin, destination): Sends a request to Google Maps Distance Matrix API to fetch traffic data.
   * process\_traffic\_data(data): Extracts distance, duration, and traffic status from the API response.
   * update\_traffic\_data(): Retrieves input values, fetches and processes traffic data, and updates the GUI with the results.
3. **Create GUI**
   * Initialize Tkinter window.
   * Load and set the background image.
   * Create and place input fields for origin and destination.
   * Create and style a button to fetch traffic data.
   * Create labels to display distance, duration, and traffic status.
4. **Run the GUI**
   * Start the Tkinter event loop.

**Detailed Explanation of the Actual Code**

1. **Library Imports**
   * requests and json are used for fetching and processing data from the Google Maps API.
   * tkinter is used to create the GUI components.
   * PIL (Pillow) is used for handling and displaying images.
2. **get\_traffic\_data(origin, destination) Function**
   * Constructs the API request URL with the provided origin, destination, and API key.
   * Sends a GET request to the Google Maps Distance Matrix API.
   * Returns the JSON response containing traffic data.
3. **process\_traffic\_data(data) Function**
   * Parses the JSON response to extract distance, duration, and traffic status.
   * Returns these values to be displayed in the GUI.
4. **update\_traffic\_data() Function**
   * Retrieves user inputs for origin and destination.
   * Calls get\_traffic\_data() to fetch the latest traffic data.
   * Uses process\_traffic\_data() to parse the response.
   * Updates the GUI labels with the fetched data.
5. **GUI Initialization**
   * A Tkinter window (root) is created, with a title "Naash Maps".
   * The background image is loaded and set for the window.
   * A frame is placed centrally to hold input fields and labels.

**Assumptions Made**

1. **API Key Validity**: It is assumed that the provided Google Maps API key is valid and has sufficient quota.
2. **Network Connectivity**: It is assumed that the user has an active internet connection to fetch data from the Google Maps API.

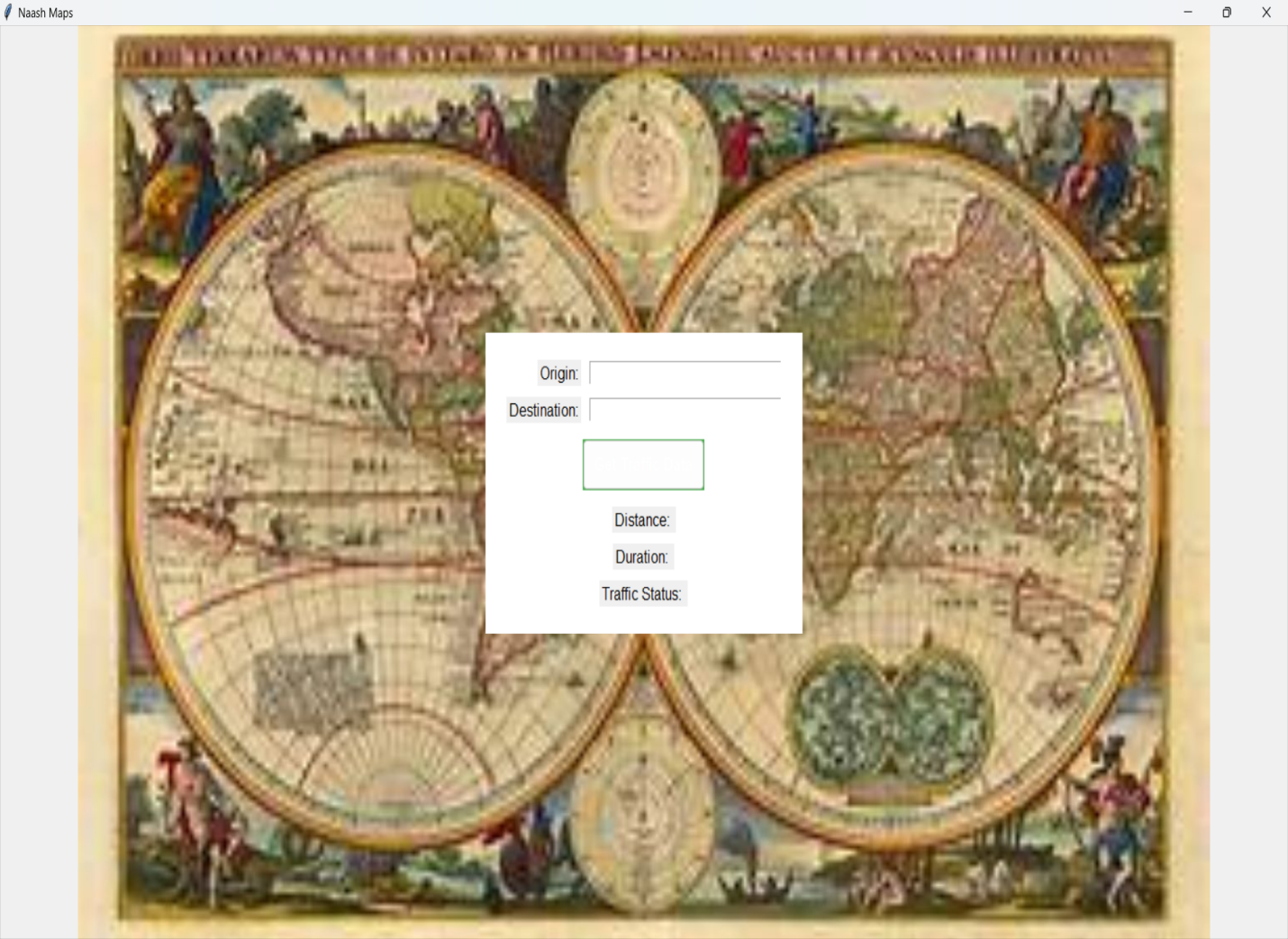
**Limitations**

1. **API Key Quota**: Google Maps API has usage limits and may require billing information. Exceeding the quota may result in errors or additional charges.
2. **Error Handling**: The code lacks comprehensive error handling for scenarios like invalid addresses or API request failures. It may need improvements to handle such cases gracefully.
3. **GUI Responsiveness**: For large-scale data or slow network conditions, the application might experience delays or become unresponsive.

**Code:**

import tkinter as tk  
from tkinter import ttk  
from PIL import Image, ImageTk  
import requests  
import json  
  
  
def get\_traffic\_data(origin, destination):  
 *"""Fetches traffic data using Google Maps Distance Matrix API"""* api\_key = "AIzaSyBc4iRkQEiUKxWs60F-HjxIvtQ\_\_a1kn4s" # Your API key  
 url = f"https://maps.googleapis.com/maps/api/distancematrix/json?origins={origin}&destinations={destination}&key={api\_key}"  
  
 response = requests.get(url)  
 data = json.loads(response.text)  
 return data  
  
  
def process\_traffic\_data(data):  
 *"""Processes the API response to extract traffic information"""* distance = data['rows'][0]['elements'][0]['distance']['text']  
 duration = data['rows'][0]['elements'][0]['duration']['text']  
 traffic\_status = data['rows'][0]['elements'][0]['status']  
 return distance, duration, traffic\_status  
  
  
def update\_traffic\_data():  
 origin = origin\_entry.get()  
 destination = destination\_entry.get()  
 traffic\_data = get\_traffic\_data(origin, destination)  
 distance, duration, traffic\_status = process\_traffic\_data(traffic\_data)  
  
 distance\_label.config(text=f"Distance: {distance}")  
 duration\_label.config(text=f"Duration: {duration}")  
 status\_label.config(text=f"Traffic Status: {traffic\_status}")  
  
  
# Create the main window  
root = tk.Tk()  
root.title("Naash Maps") # Set the window title  
  
# Load and set the background image  
bg\_image = Image.open("C:\\Users\\Avinaash.A\\Downloads\\th.jpg") # Replace with your image path  
bg\_photo = ImageTk.PhotoImage(bg\_image)  
bg\_label = tk.Label(root, image=bg\_photo)  
bg\_label.place(relwidth=1, relheight=1) # Cover the entire window  
  
# Create frames for widgets  
frame = tk.Frame(root, bg='white', padx=20, pady=20)  
frame.place(relx=0.5, rely=0.5, anchor='center') # Center the frame  
  
# Widgets for input and output  
origin\_label = tk.Label(frame, text="Origin:", font=('Arial', 12))  
origin\_label.grid(row=0, column=0, padx=5, pady=5, sticky='e')  
origin\_entry = tk.Entry(frame, font=('Arial', 12), width=25)  
origin\_entry.grid(row=0, column=1, padx=5, pady=5)  
  
destination\_label = tk.Label(frame, text="Destination:", font=('Arial', 12))  
destination\_label.grid(row=1, column=0, padx=5, pady=5, sticky='e')  
destination\_entry = tk.Entry(frame, font=('Arial', 12), width=25)  
destination\_entry.grid(row=1, column=1, padx=5, pady=5)  
  
# Aesthetic button style  
button\_style = ttk.Style()  
button\_style.configure('TButton', font=('Arial', 12, 'bold'), padding=10, relief='flat', background='#4CAF50',  
 foreground='white')  
button\_style.map('TButton', background=[('active', '#45a049')])  
  
get\_data\_button = ttk.Button(frame, text="Get Traffic Data", command=update\_traffic\_data, style='TButton')  
get\_data\_button.grid(row=2, column=0, columnspan=2, pady=10)  
  
distance\_label = tk.Label(frame, text="Distance: ", font=('Arial', 12))  
distance\_label.grid(row=3, column=0, columnspan=2, pady=5)  
  
duration\_label = tk.Label(frame, text="Duration: ", font=('Arial', 12))  
duration\_label.grid(row=4, column=0, columnspan=2, pady=5)  
  
status\_label = tk.Label(frame, text="Traffic Status: ", font=('Arial', 12))  
status\_label.grid(row=5, column=0, columnspan=2, pady=5)  
  
# Run the Tkinter event loop  
root.mainloop()

**Sample Output / Screen Shots**

****